

(19)



JAPANESE PATENT OFFICE

PATENT ABSTRACTS OF JAPAN

(11) Publication number: 2002163246 A

(43) Date of publication of application: 07.06.02

(51) Int. Cl.

G06F 17/12

G06F 15/16

(21) Application number: 2000358232

(71) Applicant: FUJITSU LTD

(22) Date of filing: 24.11.00

(72) Inventor: NAKANISHI MAKOTO

(54) PARALLEL MATRIX PROCESSING METHOD IN
SHARED MEMORY TYPE SCALAR PARALLEL
COMPUTER AND RECORDING MEDIUM

(57) Abstract:

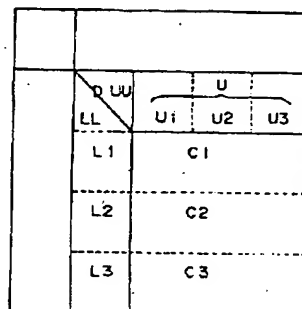
PROBLEM TO BE SOLVED: To provide a parallel matrix processing method suitable for a shared memory type scalar computer.

SOLUTION: A matrix to be LU decomposed is divided into a block D at a diagonal part and a row-direction block below the D, for example, L1 to L3. D+L1, D+L2 and D+L3 are allocated to three processors, respectively, to perform operation in parallel. The block U is updated by the method of LU decomposition, and further C1 to C3 are updated using L1 to L3 and U. The processing is conducted for the inside of the block sequentially decreased in size, thereby lastly leaving the part corresponding to the D. The D is LU decomposed by one processor to end LU decomposition for the whole matrix.

COPYRIGHT: (C)2002,JPO

本発明の実施形態に従ったLU分解の

並列処理の概念を説明する図(図1)



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2002-163246
(P2002-163246A)

(43) 公開日 平成14年6月7日(2002.6.7)

(51) Int.Cl.⁷
G 0 6 F 17/12
15/16

識別記号
6 1 0

F I
G 0 6 F 17/12
15/16

テマコード(参考)
5 B 0 4 5
6 1 0 Z 5 B 0 5 6

審査請求 未請求 請求項の数7 OL (全 12 頁)

(21) 出願番号 特願2000-358232(P2000-358232)

(22) 出願日 平成12年11月24日(2000.11.24)

(71) 出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中4丁目1番
1号

(72) 発明者 中西 誠

神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(74) 代理人 100074099

弁理士 大菅 義之 (外1名)

Fターム(参考) 5B045 AA07 DD02 DD12 GG11
5B056 AA04 BB02

(54) 【発明の名称】 共有メモリ型スカラ並列計算機における並列行列処理方法、及び記録媒体

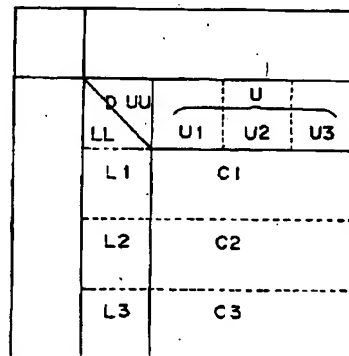
(57) 【要約】

【課題】 共有メモリ型スカラ計算機に適した並列行列処理方法を提供する。

【解決手段】 LU分解すべき行列を対角部分のブロックDとDの下側にある列方向のブロック、例えば、L1～L3に分ける。そして、3つのプロセッサのそれぞれに、D+L1、D+L2、D+L3を割り振り、並列して演算を行わせる。そして、ブロックUをLU分解の方法によって更新し、更に、L1～L3とUとを用いてC1～C3を更新する。この処理を順次小さくなっていく内側もブロックに施すことにより、最後に、Dに対応する部分だけが残る。このDを1つのプロセッサでLU分解することにより行列全体のLU分解を終える。

本発明の実施形態に従ったLU分解の

並列処理の概念を説明する図(その1)



【特許請求の範囲】

【請求項 1】複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、
 行列を小行列ブロックに分けるブロック化ステップと、
 該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、
 該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、
 該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップと、を備えることを特徴とする並列行列処理方法を情報装置に実現させるプログラムを格納した、情報装置読み取り可能な記録媒体。

【請求項 2】該行列演算は、行列の LU 分解であることを特徴とする請求項 1 に記載の記録媒体。

【請求項 3】前記複数のプロセッサモジュールが有する小行列ブロックのデータからそれぞれがピボットの候補を抽出する抽出ステップと、
 該ピボットの候補から該複数のプロセッサモジュールに共通のメモリ領域において、データ値が最大値を示すピボット候補を最終的なピボットと決定するピボット決定ステップと、を更に備え、該決定されたピボットを用いて LU 分解を行うことを特徴とする請求項 2 に記載の記録媒体。

【請求項 4】前記 LU 分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、1つのプロセッサモジュールで LU 分解することにより、該行列全体について LU 分解を完了することを特徴とする請求項 2 に記載の記録媒体。

【請求項 5】該行列演算は、行列のコレスキー分解あるいは変形コレスキー分解であることを特徴とする請求項 1 に記載の記録媒体。

【請求項 6】前記コレスキー分解あるいは変形コレスキー分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、1つのプロセッサモジュールで LU 分解することにより、該行列全体について LU 分解を完了することを特徴とする請求項 2 に記載の記録媒体。

【請求項 7】前記更新ステップにおいて、更新すべき小行列ブロックの三角行列部分を前記複数のプロセッサモジュールの数の 2 倍の数のブロックに分割し、該分割された三角行列部分のブロックを 2 つずつ組み合わせて、各プロセッサモジュールのローカルメモリ域に格納し、演算をプロセッサモジュールに行わせることを特徴とする請求項 5 に記載の記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、共有メモリ型スカラ並列計算機における並列行列処理方法に関する。

【0002】

【従来の技術】連立 1 次方程式を計算機によって解く場合には、連立 1 次方程式を行列表示し、この行列について処理を施すことによって、解の求めやすい形に変形し、このような形にしてから方程式の解を求める方法が採られている。

【0003】すなわち、連立 1 次方程式は、係数を表す行列と、変数を表す列ベクトルとの積が所定の列ベクトルに等しくなるというように記述することが出来る。ここで、LU 分解 (LU Factorization) という方法によれば、係数を表す行列を上三角行列 (upper-triangular matrix) と下三角行列 (lower-triangular matrix) に分解することによって、連立 1 次方程式の解を求める。したがって、この場合においては、係数行列を LU 分解することが連立 1 次方程式の解を得るために重要な処理となる。また、LU 分解の特別な場合として (変形) コレスキー分解 (Cholesky Factorization) という行列の分解方法がある。

a) 実行列の連立 1 次方程式の解法

実行列の連立 1 次方程式の解法に関して、ベクトル並列計算機での連立 1 次方程式は、ブロック化した外積型の LU 分解をベースに並列化を行っている。列ベクトルを何本か束ねたブロックを LU 分解 (1) した後、対応した行ベクトルを束ねたブロックを更新して (2) から、正方小行列を更新 (3) する処理を繰り返す。

【0004】従来、(1) の処理は、一つのプロセッサで逐次的に行っていた。並列効率を高めるためにブロック幅は 12 (列あるいは行: 係数行列の行幅あるいは列幅を示す) 程度の比較的小さい値としていた。この結果 (2) 及び (3) の部分の更新も幅 12 程度の行列演算となった。

【0005】最もコストの大きい (3) の計算で、幅が 12 程度と小さくても、効率の良い方法があった。共有メモリ型スカラ並列計算機 (SMP) では、幅が小さいと性能を引き出せない。これは以下の理由による。

【0006】すなわち、(3) の計算は、行列積である。幅が小さいと更新する行列の要素 (メモリに格納されている) をロードして、更新結果をストアするというメモリにアクセスするためのコストが行列の更新を行う演算に比べて大きくなり性能を引き出せない。

【0007】このため、ブロック幅を大きくする必要があるが、ブロック幅を大きくすると、このブロックの LU 分解のコストが大きくなり並列化効率が落ちる。

b) 正値対称行列の連立 1 次方程式の解法

正値対称行列の連立 1 次方程式の解法に関して、下三角行列部分のみを利用してコレスキー分解を行うときに、分散メモリ型並列計算機では小行列ブロックを `cyc1`

に各プロセッサに分散して負担させ、各プロセッサの負荷を均等にして解いていた。実行列の連立1次方程式と同じようにブロック化するときブロック幅を比較的小さくでき、並列化効率を高めることが可能であった。SMPでは、上記(3)の更新で現れる行列積でブロック幅が大きい方が性能が良いため、ブロック幅を大きくする必要がある。

【0008】

【発明が解決しようとする課題】すなわち、共有メモリ型スカラ並列計算機では、LU分解あるいはコレスキー分解における行列積による行列の更新に必要とされるコストよりも、共有メモリにアクセスするためのコストが大きくなってしまい、従来ベクトル並列計算機で行っていた方法をそのまま共有メモリ型スカラ計算機に適用しても十分な性能を引き出せない。

【0009】本発明の課題は、共有メモリ型スカラ計算機に適した並列行列処理方法を提供することである。

【0010】

【課題を解決するための手段】並列行列処理方法は、複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、行列を小行列ブロックに分けるブロック化ステップと、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップとを備えることを特徴とする。

【0011】本発明によれば、複数のプロセッサモジュールがそれぞれ有するローカルメモリ領域を有効に利用して、効率的な行列演算を行うことが出来る。

【0012】

【発明の実施の形態】図1は、共有メモリ型スカラ並列計算機のハードウェア構成例を示す図である。

【0013】共有メモリ型スカラ並列計算機は、複数のプロセッサ10-1、10-2、・・・10-nが2次キャッシュメモリ13-1、13-2、・・・13-nを介して相互結合網12に接続される。各プロセッサ10-1、10-2、・・・10-nは、その内部あるいは、2次キャッシュメモリ13-1、13-2、・・・13-nよりプロセッサ側に1次キャッシュメモリが設けられる。また、各プロセッサ10-1、10-2、・・・10-nに共有となっているメモリモジュール11-1、11-2、・・・11-nは、相互結合網12を介してプロセッサ10-1、10-2、・・・10-nがアクセス可能となっている。プロセッサ10-1、10-2、・・・10-nがデータ処理を行う場合には、まず、メモリモジュール11-1、11-2、・・・11

-nから1つのプロセッサが担当するデータを2次キャッシュメモリ13-1、13-2、・・・13-nに格納し、更に、2次キャッシュメモリから処理単位となるデータを1次キャッシュメモリにコピーして処理を行う。

【0014】処理が終わると、1次キャッシュメモリから2次キャッシュメモリに処理データが格納され、2次キャッシュメモリ内のデータが全て処理し終わると、メモリモジュール11-1、11-2、・・・11-nの内、最初にデータを持ってきたメモリモジュールに対してデータの更新を行う。また、次のデータ処理を行う場合には、上述したように、メモリモジュールから各プロセッサが担当する分のデータを2次キャッシュメモリに格納し、1次キャッシュメモリに処理単位のデータを持ってきて、プロセッサが処理を行う。このような処理を繰り返して、並列にデータ処理を完了する。このとき、各プロセッサが処理した後のデータをメモリモジュールに書き込み、次の処理のために、再びメモリモジュールからデータを読み込む際、各プロセッサが自分のタイミングでデータの読み込みを行っていたのでは、データ更新された後のデータを読み込むべきところを、データ更新される前のデータを読み込んでしまう可能性が有る。したがって、このときには、全てのプロセッサがメモリモジュールにデータ更新し終わるまで、他のプロセッサがメモリモジュールからデータを読み込まないようにする必要がある。このように、プロセッサのメモリモジュールからのデータの読み込みを制限して、全体のプロセッサの処理の同期をとることをバリア同期(Barrier Synchronization)を取るといふ。

【0015】図2及び図3は、本発明の実施形態に従ったLU分解の並列処理の概念を説明する図である。図2は、処理すべき行列の模式図であり、図3は、処理単位となるデータの構成を説明する図である。

a) 本実施形態に従った実行列の連立1次方程式の解法本実施形態においては、プロセッサが処理を担当する小行列ブロックの幅を従来より大きくしてLU分解する部分を並列化する。すなわち、図2において、従来では、L1～L3の部分は、ブロック幅を小さくして、1つのプロセッサ(PE、あるいは、スレッド)で処理していたが、本実施形態においては、このブロック幅を大きくすると共に、L1～L3をそれぞれ別のスレッドに割り当てて、並列に処理させる。なお、ここでは、スレッドの数を3つとしている。

【0016】共有メモリ型スカラ並列計算機の各プロセッサは独立に1次及び2次キャッシュが備わっている。特に1次キャッシュ上のデータに載る範囲で、計算を行うことが高性能を引き出す上で重要である。

【0017】データ量の大きな問題を多数PEで解く場合、各PEでデータを局所化して全体としてはブロックのLU分解を並列に計算し、できるだけ大きなブロック

幅を確保することが必要となる。

【0018】このために、図3に示されるように、各プロセッサでローカルに作業域を確保してL2キャッシュ（2次キャッシュ）に載る大きさで計算を行う。このとき、並列に更新するとき必要なブロック対角部分Dは、各PE（各スレッド）でコピーして各プロセッサ（各スレッド）で冗長に計算する。また、LU分解において、ピボットを取った後、行ベクトルの入れ替えを行う場合は、いずれかのプロセッサの2次キャッシュメモリ上に設けられた共用メモリ域を介して、各PEで通信して、必要な情報の共用を行うようにする。

【0019】b：ブロック幅、k：各プロセッサが分担する列ブロックの1次元目の大きさ（ブロック幅が小行列ブロックの列方向であるので、小行列ブロックの行の数、すなわち、図2におけるL1～L3のブロックの合計の行の数）としたとき、 $b \times (b+k) \times 8 \sim 8Mb$ byteを満たすものをブロック幅として採用する。

【0020】そして、作業域（1次キャッシュメモリ）にコピーした部分に関して、キャッシュメモリ上のデータを利用したLU分解を行う。また、処理すべき行列が占有するメモリ量が大きく（行列が大き）、それに比べてプロセッサ数が少ないため、並列処理向けの列ブロックのブロック幅が小さくなるときは、ブロック幅を分割して必要なブロックを確保してから、行ブロックの更新と行列積の更新を並列に行う。

【0021】更に、各プロセッサ（各スレッド）での作業域上でのLU分解は、内積法など更新部分の内積ベクトルの長さが大きな方法を、例えば、アルゴリズムの再帰的な呼び出しで、更新部分の性能を引き出しながらLU分解を行う方法を利用することで、キャッシュ上のデータを効率よく利用する。

【0022】図4は、本実施形態のLU分解の処理の流れを示す概略フローチャートである。まず、ステップS1において、スレッド数及び問題の大きさ（処理すべき行列の大きさ）からブロック幅を決定する。次に、ステップS2において、各スレッドが処理するブロックを決定し、各スレッドで処理するブロック（図2のD及びLi）を作業域にコピーする。そして、ステップS3において、各スレッドでピボットを決定し、その中での最大値を示すピボットを共用域を使って決定し、最大値を示すピボットを用いて、行ベクトルを入れ替え、各スレッドで、上記ブロックDとブロックLiとをLU分解する。

【0023】ステップS4においては、処理が終わりか否かを判断し、終わりの場合には、処理を終了するが、終わりでない場合には、ステップS5において、各スレッドで並列にブロックLL（図2参照）を使って、ブロックUi（図2参照）をLU分解のアルゴリズムに従って更新する。そして、ステップS6において、各スレッドで並列に、ブロックCi（図2参照）をブロックLi

とブロックU（Uiを組み合わせたもの：図2参照）の積で更新する。そして、ステップS2に戻り、次のブロック（Ciからなるブロック）を、同様の方法でLU分解する。そして、次第に小さくなる未処理ブロックが最後にブロックDに対応する部分のみになり、1つのスレッドでLU分解が完了すると、行列全体についてLU分解が終了する。

【0024】図5～図10は、本実施形態のLU分解の方法をより詳細に説明する図である。ここでは、 2048×2048 の行列を4スレッドでLU分解する場合を例にとって説明する。

【0025】まず、 2048×2048 の行列をブロック幅256でLU分解するものとする。各ブロックの区分けは、図5に示すとおりである。そして、4つのCPU（スレッド）で処理を実行する場合、各スレッドで連続な領域（ $(256+448) \times 256 : 8MB$ （L2キャッシュの大きさ）より小さい領域）を確保し、図6に示すように、各スレッドの各領域にD1+L1、D1+L2、D1+L3、D1+L4をコピーする。

【0026】なお、ブロック幅は例えば、以下のようにして決める。問題の大きさ（行列の大きさ：行列の次数）をn、スレッドの数を#THREADとして、

【0027】

【数1】

$$\sqrt[3]{(n^3 / \# \text{THREAD}) \times \frac{1}{100}} = nb$$

【0028】とにおいて、

$nb \geq 512$ なら、ブロック幅=512

$nb \geq 256$ なら、ブロック幅=256

$nb \geq 128$ なら、ブロック幅=128

それ以外、ブロック幅=64

というように、メニュー化しておき、この中から選ぶようにする。

【0029】すなわち、LU分解のコストは、 $2n^3 / 3$ （nは行列の次数）で、 n^3 に比例する。したがって、全体のコストを#THREADで並列化し、その1%が最後に1スレッドで行うブロック幅程度となるように決める。

【0030】ここで、理解を助けるため、並列化しない場合のLU分解のアルゴリズムを図7に示す。図7においては、 $LT = D1 + L1 + L2 + L3 + L4$ の部分をLU分解するアルゴリズムを示している。LTは、 2048×256 のブロックとなっている。

【0031】まず、(1)の部分において、ピボットを決定する。iblkはLTの幅であり、今の場合、256である。また、lengは、LTの長さであり、今の場合、2048である。jjには、ピボットの存在する行番号が、TMPには、ピボットの絶対値が設定される。

【0032】そして、(2)の部分において、現在処理しているLT内の列番号iより、ピボットの存在する行

番号 j が大きい場合に、 i 番の行のデータを j 番の行のデータと入れ替える。次に、(3)の部分で、列 i のピボットを使って、LU分解の演算を行う。

【0033】この(1)～(3)を i が $1 \sim \text{iblk}$ にわたって繰り返し演算する。ここで、LTの長さである、 leng がもっと大きくなると、これらの処理はL2キャッシュメモリのデータを入れ替えてしまい、著しい性能低下を引き起こす。そこで、図5のように、データを分散し、L2キャッシュメモリにデータを保持したまま処理を行うようにする。各スレッドにおけるアルゴリズムは図8に示すとおりである。

【0034】なお、図8においては、LT i は、ローカル域、 $\text{pivot}(4)$ 、 GPivot 、 $\text{ROW}(\text{iblk})$ は、共用域に格納されるデータである。まず、(4)において、各スレッドでピボットを取る。そして、(5)で最大値を配列 $\text{pivot}(4)$ のスレッド番号の要素に格納する。(5)の後に、バリア同期を取って、(6)において、最大ピボットを持つスレッド番号を GPivot に格納する。そして、(6)の後で、再びバリア同期を取る。次に、(7)において、最大ピボットを持つスレッドが共用域 ROW に最大ピボットの行ベクトルを格納し、バリア同期を取る。

(8)においては、 GPivot が0のときは、最大ピボットは $D1$ の内にあるか、入れ替え不要であり、ローカル域に入れ替える。 GPivot が $\# \text{THREAD}$ に等しいとき、すなわち、 GPivot が0より大きい場合には、最大ピボットを持たないスレッドは、 ROW の内容と i 行目の行ベクトルを入れ替える。そして、(9)、(10)において、LU分解のための演算を行う。そして、上記(4)～(10)を処理するブロックの全ての列について行う。すなわち、 $1 \sim \text{iblk}$ までの i について処理を繰り返す。

【0035】ここで、LT i のLU分解の最後でバリア同期を取る。そのあと、各スレッドの $D1$ の部分は L と U にLU分解されている。そして、各スレッドで $U1 \leftarrow LL^{-1}U1$ 、 $U2 \leftarrow LL^{-1}U2$ 、 $U3 \leftarrow LL^{-1}U3$ 、 $U4 \leftarrow LL^{-1}U4$ を各スレッドで並列に計算する。この計算の後で、 $D1$ 、 $L1$ 、 $L2$ 、 $L3$ 、 $L4$ をローカル域から行列 A にコピーバックし、バリア同期を取る。更に、 $C1 \leftarrow C1 - L1 \times U$ 、 $C2 \leftarrow C2 - L2 \times U$ 、 $C3 \leftarrow C3 - L3 \times U$ 、 $C4 \leftarrow C4 - L4 \times U$ を並列に書くスレッドで行い、最後にバリア同期を取る。

【0036】図9は、上記処理によって、1段階の処理が終わった後の行列の様子を説明する図である。図9に示されるように、上記処理をすることによって、行列の外側の行及び列が処理されたので、次に、残された左下の部分を同様の方法によって順次処理する。すなわち、ブロック幅 iblk を縮小した部分を同じように分割し、図9に示すように、 $D1$ 、 $L1$ 、 $L2$ 、 $L3$ 、 $L4$ のブロックに分けて、各スレッドにコピーし、上記と同じ処理を行う。このように、処理を繰り返していくと、最後

に 256×256 のブロックが残る。この部分は1つのスレッドでLU分解して処理を終了する。

【0037】なお、上記処理では、LT i をLU分解するとき、キャッシュメモリ上のデータを効率よく利用するため再帰的なLU分解を利用している。また、 $C_i \leftarrow C_i - L_i \times U$ の演算は、キャッシュメモリ上のデータを有効に利用した方法が既知の技術として知られている。

【0038】図10は、再帰的LU分解アルゴリズムを説明する図である。再帰的LU分解のアルゴリズムはサブルーチンLUとして与えられる。LUの取る変数は、図9のアルゴリズムで出てきた、LT i (各スレッドで $D1 + L_i$ を格納)、 k (LT i の1次元目の大きさ)、 iblk (ブロック幅)の他に、LU分解を始める位置を示す ist 、LU分解を行う幅である nwid である。

【0039】まず、サブルーチンの最初で、 nwid が8、すなわち、LU分解を行う幅が8であるか否かを判断する。YESの場合には、LT i ($\text{ist} : k$ 、 $\text{ist} : \text{ist} + \text{nwid} - 1$) を並列にLU分解する。ここで、 $\text{ist} : k$ と言う表記は、変数が ist から k までのLT i を示す意味で、 $\text{ist} : \text{ist} + \text{nwid} - 1$ というのは、変数が ist から $\text{ist} + \text{nwid} - 1$ までのLT i を示す意味である。以下においても同様である。

【0040】LT i のLU分解においては、上記(4)～(10)の処理を行う。ただし、行の入れ替え部分は、長さ iblk で、LT i ($i, 1 : \text{iblk}$) を入れ替える。また、上記判断がNOの場合には、LU分解を行う幅 nwid を2で割った値のLU分解をのLU分解のサブルーチンを再帰的に呼び出して行う。その後、TRSというルーチンを呼び出す。このルーチンは、LT i ($\text{ist} : \text{ist} + \text{nwid} / 2 - 1$ 、 $\text{ist} + \text{nwid} / 2 : \text{ist} + \text{nwid}$) を更新する。更に、LT i ($\text{ist} : \text{ist} + \text{nwid} / 2 - 1$ 、 $\text{ist} : \text{ist} + \text{nwid} / 2 - 1$) の下三角行列 L を利用して、 LL^{-1} を C_i に左からかけて更新する。次に、MMというルーチンを呼び出す。このルーチンでは、

$LT_i(\text{ist} + \text{nwid} / 2 : k, \text{ist} + \text{nwid} / 2 : \text{ist} + \text{nwid}) = LT_i(\text{ist} + \text{nwid} / 2 : k, \text{ist} + \text{nwid} / 2 : \text{ist} + \text{nwid}) - LT_i(\text{ist} + \text{nwid} / 2 : k, \text{ist} : \text{ist} + \text{nwid} / 2 - 1) \times LT_i(\text{ist} : \text{ist} + \text{nwid} / 2 - 1, \text{ist} + \text{nwid} / 2 : \text{ist} + \text{nwid})$

を演算する。そして、その後、バリア同期を取り、LU分解のサブルーチンを再帰的に呼び出し、処理した後、処理が終わると、サブルーチンを抜ける。

b) 正値対称行列の連立1次方程式の解法

図11～図13は、正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図である。

【0041】実行列の連立1次方程式と同様に、行列を D 、 $L1$ 、 $L2$ 、 $L3$ に分割して、各スレッドに対角行列 D と更新する列ブロック部分 $L1$ 、 $L2$ 、 $L3$ を作業

域にコピーして(図12参照)、独立に並列にして列ブロック部分をコレスキー分解する。なお、この場合、ピボットを取る必要がない。この分解された列ブロックを利用して、小下三角行列($C1 \sim C6$ からなる)を更新する。この更新部分を、並列に行うとき負荷を均等にするために、更新するべき下三角行列をスレッド数を $\#T$ としたとき、 $2 \times \#T$ に同じブロック幅に分ける(すなわち、 $C1$ と $C6$ 、 $C2$ と $C5$ 、 $C3$ と $C4$ を組み合わせ処理を行うようにする)。各スレッドは i 番目及び $2 \times \#T + 1 - i$ 番目のブロックを更新することで負荷を均等にする。

【0042】現在考えている行列が正値対称行列であるので、図2のUに対応する部分は、 $L1 + L2 + L3$ からなる列ブロックの転置 L^T となっているので、この場合には、 L^T 部分は、演算を行う必要が無く、 $L1$ 、 $L2$ 、 $L3$ をそれぞれ転置してコピーすればよい。

【0043】図13は、再帰的コレスキー分解の処理の進行の状況を説明する図である。まず、(1)において、行列の一番左のブロックをコレスキー分解し、30の部分をも31にコピーする。そして、31の下で点線で囲まれている部分を斜線部分を用いて更新する。次に、

(2)において、処理する列の幅を2倍にして32の部分をも33にコピーし、33の下で点線で囲まれた部分を斜線部分を用いて、更新する。そして、(3)に進んで、34の部分をも35にコピーし、35の下で点線で示された部分を、斜線部分を用いて更新する。更に、

(4)において、36で示される部分を37にコピーし、37の下で点線を斜線部分を用いて更新する。更に、(5)において、38を39にコピーし、39の下で点線で示されている部分を斜線部分を用いて更新し、

(6)において、40を41にコピーし、斜線部分を用いて、41の下で点線を斜線部分を用いて更新する。更に、(7)において、42の部分をも43にコピーし、斜線部分を用いて43の下で点線を斜線部分を用いて更新する。このように、行列の一部にコレスキー分解を再帰的に繰り返し適用しながら、最終的には行列全体をコレスキー分解する。

【0044】図14～図16は、変形コレスキー分解のアルゴリズムをより詳細に説明する図である。ここでは、説明を簡略化するため4スレッドを使って処理を行う場合を説明する。

【0045】まず、各スレッドに図14に示される $D \times$ 及び L_i を連続的な領域 LT_i にコピーする。そして、 LT_i を LDL^T 分解する。 LDL^T 分解は再帰的な方法で行う。そして、 DL_i 以下の計算で値を並列にコピーする。

【0046】 $DL_i \leftarrow D \times L_i^T$ 、ここで、 D は $D \times$ の対角要素であり、また、右辺は各スレッドのローカル領域。そして、 $D \times$ (スレッド1番から)他の L_i は並列に、もとの領域にコピーバックする。そして、ここで、バリア同期を取り、図15に示されるように、 $C1$ と C

8、 $C2$ と $C7$ 、 $C3$ と $C6$ 、 $C4$ と $C5$ をペアとして、各スレッドで並列に更新する。すなわち、

・スレッド1では、

$$C1 \leftarrow C1 - L11 \times DL11$$

$$C8 \leftarrow C8 - L42 \times DL42$$

・スレッド2では、

$$C2 \leftarrow C2 - L12 \times DL12$$

$$C7 \leftarrow C7 - L41 \times DL41$$

・スレッド3では、

$$C3 \leftarrow C3 - L21 \times DL21$$

$$C6 \leftarrow C6 - L32 \times DL32$$

・スレッド4では、

$$C4 \leftarrow C4 - L22 \times DL22$$

$$C5 \leftarrow C5 - L31 \times DL31$$

という演算を行う。

【0047】そして、ここまでの演算が終わった時点でバリア同期を取り、1周り小さくなった行列の領域に関して同じ処理を行う。以上を繰り返し、最後は1つのスレッドで、 LDL^T 分解して処理を終了する。

【0048】上記、各スレッドで行う C_i の更新処理をより一般的な言葉で述べると、スレッド数を $\#THREAD$ として、 L を $2 \times \#THREAD$ 個に分割し、 C の下三角部分も同じく $2 \times \#THREAD$ 個に分割する。そして、上と下から $\#THREAD$ 個のペアを作り、このペアで C の分割した部分を更新するという処理になる。

【0049】図16は、 LDL^T 分解の再帰的アルゴリズムを示す図である。 LDL^T 分解のアルゴリズムは、サブルーチン LDL として実現される。サブルーチンが取る変数は、前述のLU分解の場合と同様である。

【0050】まず、 $nwid$ が8の場合には、(20)の部分で、直接 LDL^T 分解を行う。そして、 LT_i ($ist + 8 : k$, $ist : ist + 7$)を更新する。このとき、 LT_i ($ist : ist + 7$, $ist : ist + 7$)の上三角部分に DL^T が入っているので、 $(DL^T)^{-1}$ を右からかけることによって更新する。

【0051】(20)の最初のIF文で、 $nwid$ が8でないと判断された場合には、 $nwid/2$ を新たな $nwid$ としてサブルーチン LDL を呼び出し、実行する。ここで、 LT_i ($ist : ist + nwid/2 - 1$, $ist + nwid/2 : ist + nwid - 1$)に DL^T をコピーする。 D は、 LT_i ($ist : ist + nwid/2 - 1$, $ist : ist + nwid/2 - 1$)の対角要素であり、 L は、 LT_i ($ist + nwid/2 : ist + nwid - 1$, $ist : ist + nwid/2 - 1$)であり、この L を転置する。

【0052】そして、 LT_i ($ist + nwid/2 : k$, $ist + nwid/2 : ist + nwid - 1$)を更新する。すなわち、

$$LT_i(ist + nwid/2 : k, ist + nwid/2 : ist + nwid - 1) = LT_i(ist + nwid/2 : k, ist + nwid$$

$\text{ist} + \text{nwid} - 1) - \text{LTI}(\text{ist} + \text{nwid} / 2 : \text{k}, \text{ist} : \text{ist} + \text{nwid} - 1) \times \text{LTI}(\text{ist} : \text{ist} + \text{nwid} / 2 - 1, \text{ist} + \text{nwid} / 2 : \text{ist} + \text{nwid} - 1)$ の演算を行う。

【0053】次に、 LDL^T 分解のサブルーチン LDL を再帰的に呼び出す。そして、処理が終わったら、サブルーチンを抜ける。なお、本発明の実施形態は、上記説明から分かるように、共有メモリ型スカラ並列計算機のアルゴリズムとして与えられるので、このアルゴリズムをプログラムとして実現することになる。あるいは、該並列計算機をLU分解専用機あるいはコレスキー分解専用機として使用する場合には、ROMなどにプログラムを書き込んでおくことも可能であるが、汎用の並列計算機として使用する場合には、本発明の実施形態のアルゴリズムは、CD-ROM等の可搬記録媒体や、ハードディスクなどの記録媒体にプログラムとして記録しておき、必要に応じて、プログラムをプロセッサにロードして使用する形態が考えられる。

【0054】このような場合、本発明の実施形態のアルゴリズムを実現するプログラムは、可搬記録媒体などを使って、ユーザに配布が可能である。

(参考文献)

・LU分解の文献は、1)、2)、変形コレスキー分解の文献は2)

1) P. AMESTOY, M. DAYDE, and I. DUFF, "Use of computational kernels in the solution of full and sparse linear equations", M. COSNARD, Y. ROBERT, Q. QUINTON, and M. RAYNAL, PARALLEL & DISTRIBUTED ALGORITHMS, North-Holland, 1989, pp. 13-19

2) G. H. Golub, C. F. van Loan, "Matrix Computations", second edition, The Johns Hopkins University Press, 1989

・日本語の文献では、以下のものにLU分解及び LDL^T 分解の解説がある。

・「数値解析」森 正武著、共立出版会社

・「スーパーコンピュータとプログラミング」島崎真昭著、共立出版社

(付記1) 複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、行列を小行列ブロックに分けるブロック化ステップと、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップと、を備えることを特徴とする並列行列処理方法を情報装置に実現させるプログ

ラムを格納した、情報装置読み取り可能な記録媒体。

【0055】(付記2) 該行列演算は、行列のLU分解であることを特徴とする付記1に記載の記録媒体。

(付記3) 前記複数のプロセッサモジュールが有する小行列ブロックのデータからそれぞれがピボットの候補を抽出する抽出ステップと、該ピボットの候補から該複数のプロセッサモジュールに共通のメモリ領域において、データ値が最大値を示すピボット候補を最終的なピボットと決定するピボット決定ステップと、を更に備え、該決定されたピボットを用いてLU分解を行うことを特徴とする付記2に記載の記録媒体。

【0056】(付記4) 前記LU分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、1つのプロセッサモジュールでLU分解することにより、該行列全体についてLU分解を完了することを特徴とする付記2に記載の記録媒体。

【0057】(付記5) 該行列演算は、行列のコレスキー分解あるいは変形コレスキー分解であることを特徴とする付記1に記載の記録媒体。

(付記6) 前記コレスキー分解あるいは変形コレスキー分解は、前記行列の外側から再帰的なアルゴリズムによって順次行列の更新を行い、前記行列の内、最後に更新し残った部分を、1つのプロセッサモジュールでLU分解することにより、該行列全体についてLU分解を完了することを特徴とする付記2に記載の記録媒体。

【0058】(付記7) 前記更新ステップにおいて、更新すべき小行列ブロックの三角行列部分を前記複数のプロセッサモジュールの数の2倍の数のブロックに分割し、該分割された三角行列部分のブロックを2つずつ組み合わせて、各プロセッサモジュールのローカルメモリ領域に格納し、演算をプロセッサモジュールに行わせることを特徴とする付記5に記載の記録媒体。

【0059】(付記8) 複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機の行列演算において、行列を小行列ブロックに分けるブロック化ステップと、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメモリ領域に格納する格納ステップと、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算ステップと、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新ステップと、を備えることを特徴とする並列行列処理方法。

【0060】(付記9) 複数のプロセッサモジュールを持つ共有メモリ型スカラ並列計算機において、行列を小行列ブロックに分けるブロック化手段と、該小行列ブロックの内、対角ブロックと対角ブロックでない小行列ブロックとを該複数のプロセッサモジュールのローカルメ

メモリ領域に格納する格納手段と、該複数のプロセッサモジュールが並列に、それぞれ有するブロックを演算することにより、複数のプロセッサモジュールで、対角ブロックを冗長に演算する演算手段と、該演算ステップで得られた小行列ブロックの演算結果を使って、該行列を更新する更新手段と、を備えることを特徴とする並列行列処理装置。

【0061】

【発明の効果】本発明によれば、高性能かつスケーラビリティのある行列の処理方法が得られる。

【図面の簡単な説明】

【図1】共有メモリ型スカラ並列計算機のハードウェア構成例を示す図である。

【図2】本発明の実施形態に従ったLU分解の並列処理の概念を説明する図（その1）である。

【図3】本発明の実施形態に従ったLU分解の並列処理の概念を説明する図（その2）である。

【図4】本実施形態のLU分解の処理の流れを示す概略フローチャートである。

【図5】本実施形態のLU分解の方法をより詳細に説明する図（その1）である。

【図6】本実施形態のLU分解の方法をより詳細に説明する図（その2）である。

【図7】本実施形態のLU分解の方法をより詳細に説明

する図（その3）である。

【図8】本実施形態のLU分解の方法をより詳細に説明する図（その4）である。

【図9】本実施形態のLU分解の方法をより詳細に説明する図（その5）である。

【図10】本実施形態のLU分解の方法をより詳細に説明する図（その6）である。

【図11】正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その1）である。

【図12】正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その2）である。

【図13】正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図（その3）である。

【図14】変形コレスキー分解のアルゴリズムをより詳細に説明する図（その1）である。

【図15】変形コレスキー分解のアルゴリズムをより詳細に説明する図（その2）である。

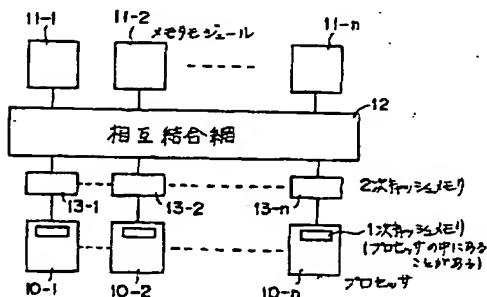
【図16】変形コレスキー分解のアルゴリズムをより詳細に説明する図（その3）である。

【符号の説明】

10-1～10-n プロセッサ
11-1～11-n メモリモジュール
12 相互結合網
13-1～13-n 2次キャッシュメモリ

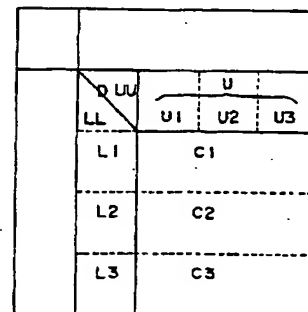
【図1】

共有メモリ型スカラ並列計算機ハードウェア
構成例を示す図



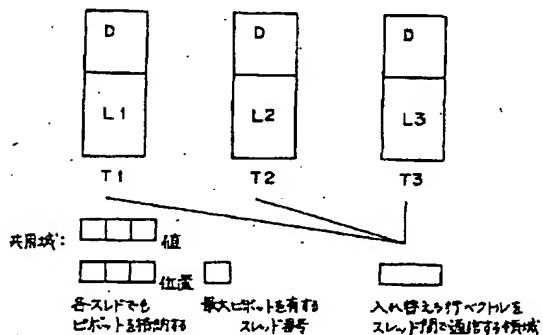
【図2】

本発明の実施形態に従ったLU分解の
並列処理の概念を説明する図（その1）



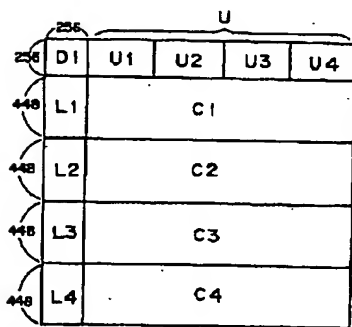
【図3】

本発明の実施形態に従ったLU分解の並列処理
の概念を説明する図(その2)



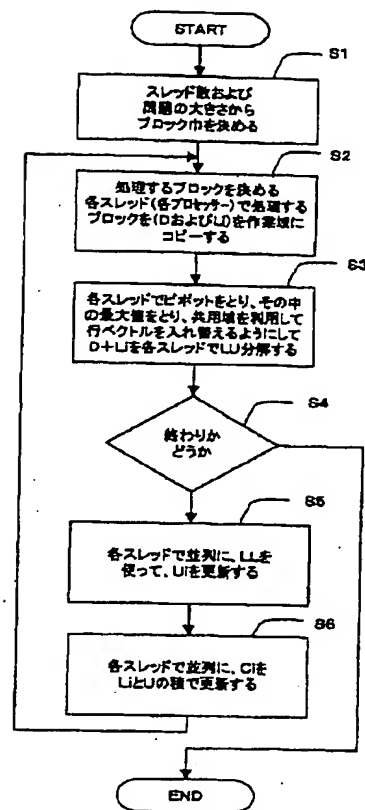
【図5】

本実施形態のLU分解の方法をより詳細に
説明する図(その1)



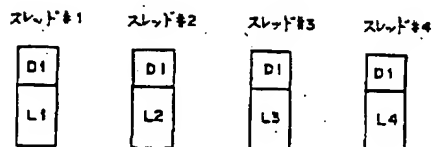
【図4】

本実施形態のLU分解の処理の流れを
示す概略フローチャート



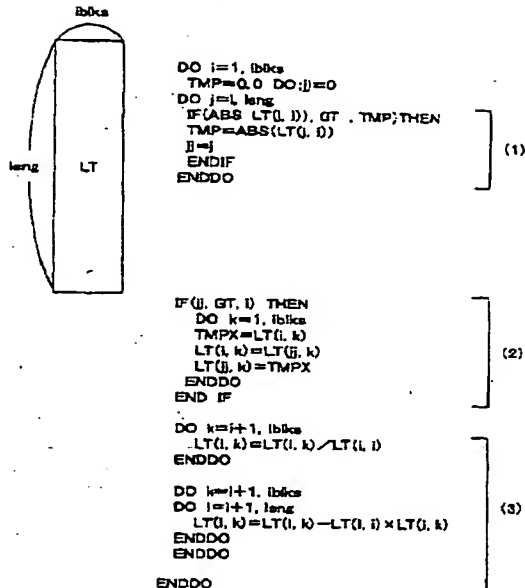
【図6】

本実施形態のLU分解の方法をより詳細に
説明する図(その2)



【圖7】

本実施形態のLU分解の方法をより詳細に説明する図(その3)



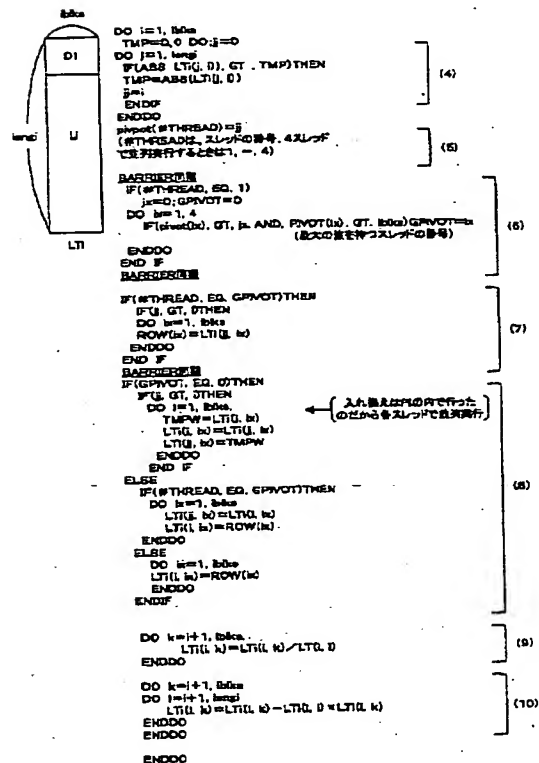
【图9】

本実施形態のLU分解の方法をより詳細に
説明する図(その5)

256	D1	U1	U2	U3	U4
384	L1	C1			
384	L2	C2			
384	L3	C3			
384	L4	C4			

【图8】

本実施形態のLU分解の方法をより詳細に説明する図(その4)



【图 10】

本実施形態のLU分解の方法をより詳細に説明する図(その6)

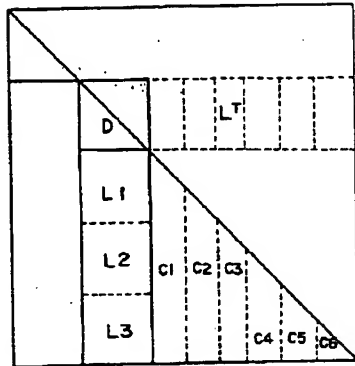
```

subroutine LU(LTI, k, lbkx, ist, nwid)
  (LTIは各スレッドでD1+Lに格納。
   k: LTIの1次元目の大きさを。
   lbkx: ブロック巾
   ist: LU分解を始める位置。
   nwid: LU分解を行う巾)
  IF(nwid, eq, 8) THEN (D1が最小)
    LTI(ist: k, ist: ist+nwid-1)を左列にLU分解する。
    ここで、(4)-(10)を行う。
    ただし行の入れ換え部分は長さbksで、LTI(1, lbkx)を入れ替える
  else
    call LU(LTI, k, lbkx, ist, nwid/2)
    call TRB(
      LTI(ist: ist+nwid/2-1, ist+nwid/2: ist+nwid)
      を更新する。LTI(ist: ist+nwid/2-1, ist: ist+nwid/2-1)
    の下三角行列Lを利用して、LL*を左から掛けて更新
  )
  call MNV(
    LTI(ist+nwid/2: k, ist+nwid/2: ist+nwid)
    =LTI(ist+nwid/2: k, ist+nwid/2: ist+nwid)
    -LTI(ist+nwid/2: k, ist: ist+nwid/2-1) x
    LTI(ist: ist+nwid/2-1, ist+nwid/2: ist+nwid)
  )
end
end subroutine

```

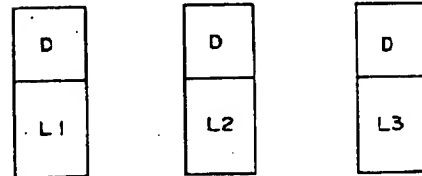
【図11】

正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図(その1)



【図12】

正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図(その2)

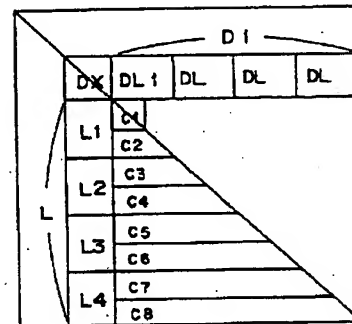
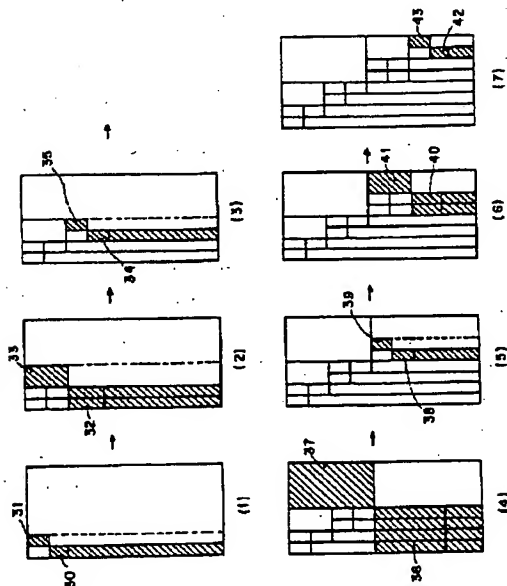


【図14】

変形コレスキー分解のアルゴリズムをより詳細に説明する図(その1)

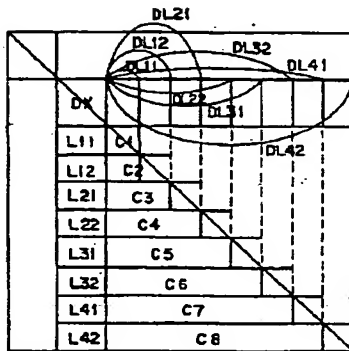
【図13】

正値対称行列の場合にコレスキー分解を行う処理の概念を説明する図(その3)



【図 15】

変形コレスキー分解のアルゴリズムをより詳細
に説明する図(その2)



【図 16】

変形コレスキー分解のアルゴリズムを
より詳細に説明する図(その3)

```

subroutine LTD(LT, k, lbks, ist, nwid)
  IF(nwid, EQ, 8) THEN (巾8が最小)
    DOI=ist, ist+7
    DOJ=I+1, ist+7
      LTI(I, J)=LTI(J, I)
      LTI(J, I)=LTI(J, I)/LTI(I, I)
    ENDDO
    DO J=I+1, ist+7
      DO I=J, ist+7
        LTI(I, J)=LTI(I, J)-LTI(I, I)*LTI(I, J)
      ENDDO
    ENDDO
    LTI(LTI(ist+8:k, ist:ist+7)を更新
    LTI(LTI(ist:ist+7, ist:ist+7)の上三角にDL2が入っているので
    (PL2)-1を右からかけて更新する
  ELSE
    call LDL(LT, k, lbks, ist, nwid/2)
    LTI(ist:ist+nwid/2-1, ist+nwid/2:ist+nwid-1)
    LTI(ist+nwid/2-1, ist+nwid/2:ist+nwid-1)
    LTI(ist+nwid/2:ist+nwid-1, ist:ist+nwid/2-1)
    LTI(ist+nwid/2:ist+nwid-1, ist:ist+nwid/2-1)
    このL2を拡張する
    LTI(ist+nwid/2:k, ist+nwid/2:ist+nwid-1)を更新
    LTI(ist+nwid/2:k, ist+nwid/2:ist+nwid-1)
    LTI(ist+nwid/2:k, ist:ist+nwid/2-1)
    LTI(ist:ist+nwid/2-1, ist+nwid/2:ist+nwid-1)
    CALL LDL(LT, k, lbks, ist+nwid/2, nwid/2)
  ENDIF
  RETURN
END

```